Empirical Dynamic Modeling Analysis Applied to EEG Engagement Index and Recall Scores

Ethel Pruss, Anita Vrins, Caterina Ceccato, Jos Prinsen

28-04-2023

Introduction

In this entry, we will explore the presumably non-linear relationship between EEG data collected during 10 minutes of a listening task where medical information was provided by a social robot and the participant's ability to recall that information. To that end, we have created epochs in the data for time windows when certain information was given and linked them to recall test scores, where participants will have either a "1" if they correctly answered a question with information from that window or a "0" if they answered incorrectly.

As in the previous entries, we will be using 8-channel EEG data (standard 10/20 electrode placement), although in this module we will be working with the data of 38 participants. 2 participants had to be removed from this analysis due to lost metadata that made linking up recall scores impossible. The data has already been cleaned and preprocessed (the preprocessing steps and more information on the dataset can be found here).

Empirical Dynamic Modeling Approach

Empirical Dynamic Modeling (EDM) is a group of techniques used to study complex systems that don't follow linear patterns. It doesn't rely on having specific equations to describe how the system works. Instead, it reconstructs the system's behavior by looking at the patterns in its state over time. This is done by creating a "shadow" version of the system called an attractor, which keeps the same overall structure and properties. The neighborhoods and paths in this reconstructed version match those of the original system, allowing us to understand its dynamics. The approach is based on Taken's theorem (1981). In simpler terms, EDM helps us understand complex systems by observing their patterns and reconstructing their behavior without needing to know all the rules upfront.

EEG data and participant's cognitive processing of information (such that they do or do not successfully store it in memory fully enough to be retrieved later) could be an example of such a complex system. We have previously tried to look for linear correlations between these variables but could not find any. However, EDM can help us explore non-linear relationships between two time series. One complication is that the recall data is a binary time series, which is not typical for this type of analysis. However, I did find a few papers that worked with binary or categorical data and convergent cross mapping to some extent

(Berkel 2020; Sethi & Mittal 2022). These were primarily studies generating features for machine learning algorithms and none were directly comparable to what we want to do but perhaps enough to give some indication that convergent cross mapping could still be possible if one of the signals is binary.

Steps We Need to Take Prior to EDM

According to Chang et al. (2017), the following preprocessing steps are needed for successful EDM analysis:

- The data needs to be sampled at equal intervals (check)
- The data needs to be normalized to zero mean and unit variance (this still needs to be done)
- The data needs to be stationary (we need to check if it is stationary and detrend if neccessary)

In the first section, we will adress these preprocessing steps.

First, let's get some packages we'll be using ready:

```
knitr::opts_chunk$set(echo = TRUE)
require(rEDM)

## Loading required package: rEDM

require(Kendall)

## Loading required package: Kendall

require(nonlinearTseries)

## Loading required package: nonlinearTseries

## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo

##
## Attaching package: 'nonlinearTseries'

## The following object is masked from 'package:grDevices':
##
##     contourLines

require(ggplot2)

## Loading required package: ggplot2

require(tidyverse)

## Loading required package: tidyverse
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse
2.0.0 ──
## ✓ dplyr     1.1.1     ✓ readr     2.1.4
## ✓ forcats   1.0.0     ✓ stringr   1.5.0
## ✓ lubridate 1.9.2     ✓ tibble    3.2.1
## ✓ purrr     1.0.1     ✓ tidyr     1.3.0
## ── Conflicts ────────────────────────────────────────
tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
conflicts to become errors

require(broom)

## Loading required package: broom

require(knitr)

## Loading required package: knitr

options(scipen=999) # Just for better readability
```

The sources for these are included in the references section at the bottom!

Preprocessing

1. Let's load the data, extract the relevant columns and generate a timeseries out of them.

```
combined_data <- read.csv("~/Complex Project/combined_data.csv")

df <- subset(combined_data, Participant == 5 & AdaptiveRandom == 1) #
Selecting a random participant
Time <- 1:length(df$Fz) # Note this is a simple index. Time is actually
sampled at 256 Hz
df <- cbind(Time,df)
df$MeanEEG <- rowMeans(df[,1:8], na.rm=TRUE)
head(df)

##       Time         Fz         C3         Cz          C4        Pz
PO7
## 43552    1 -30.429989  4.6413068 -15.117018   1.4301815  3.559436
17.769739
## 43567    2  26.270626 14.8062692   1.779232   4.0303910 -3.764754  -
9.154879
## 43591    3   0.240387  3.2308940  28.468729 -13.3363069 -1.665471
1.157088
## 43604    4  14.557969 -2.3907941  29.412638  -7.9933855 -2.330357  -
7.739072
```

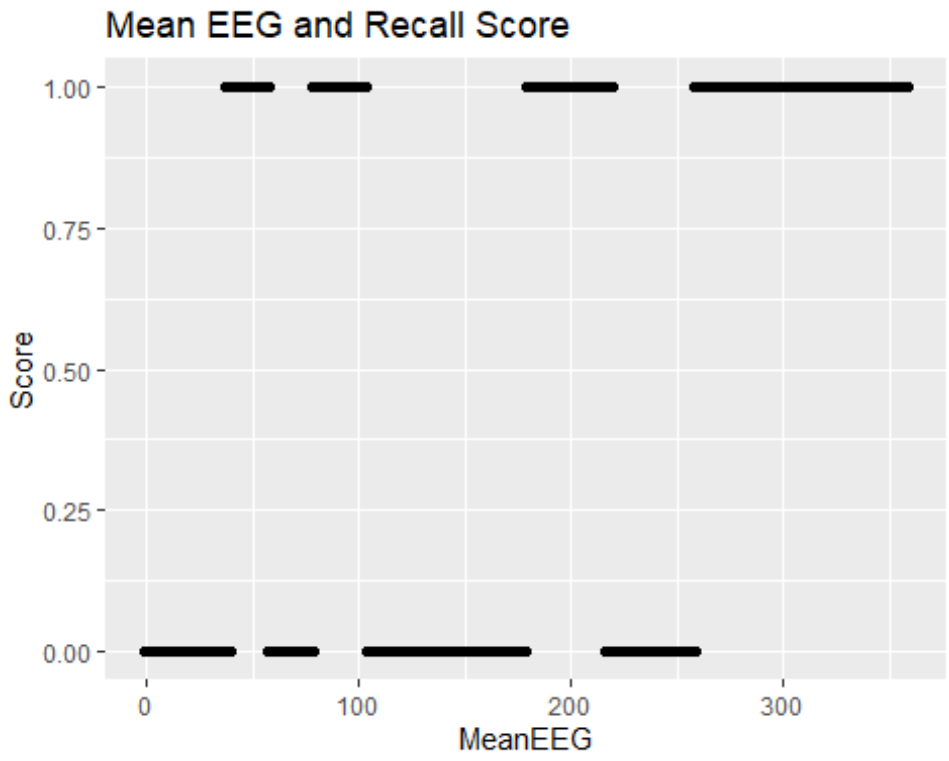
```
## 43606    5  23.951332  0.2889619  24.637549  -0.1686617 -4.183120 -
23.218889
## 43623    6  43.981476 14.1901694  -5.511497  13.1641660 -7.712878 -
20.723576
##                Oz        PO8 AdaptiveRandom FirstSecond Participant Score
## 43552  11.316693   6.829650              1           0           5     0
## 43567 -18.195001 -15.771884              1           0           5     0
## 43591  -3.038505 -15.056815              1           0           5     0
## 43604 -15.774603  -7.742395              1           0           5     0
## 43606 -18.114260  -3.192912              1           0           5     0
## 43623 -24.538092 -12.849768              1           0           5     0
##          MeanEEG
## 43552 -0.7287063
## 43567  2.2214856
## 43591  2.2571019
## 43604  1.4677994
## 43606  1.0241141
## 43623  2.3562210
```

Let's take a look at the time series.

```
plot1 <- ggplot(data=df, aes(x=Time, y=MeanEEG)) + geom_line(aes(Time,
MeanEEG),color="grey") + geom_line(aes(Time,Score), color="red") +
xlab("Time") + ylab("Value") + ggtitle("Mean EEG-Grey and Score-Red Time
Series")
plot1
```

Now let's look at a scatterplot.

```
plot2 <- ggplot(data=df, aes(x=MeanEEG, y=Score)) + geom_point()
+ggtitle("Mean EEG and Recall Score")
plot2
```



Pre-processing the data

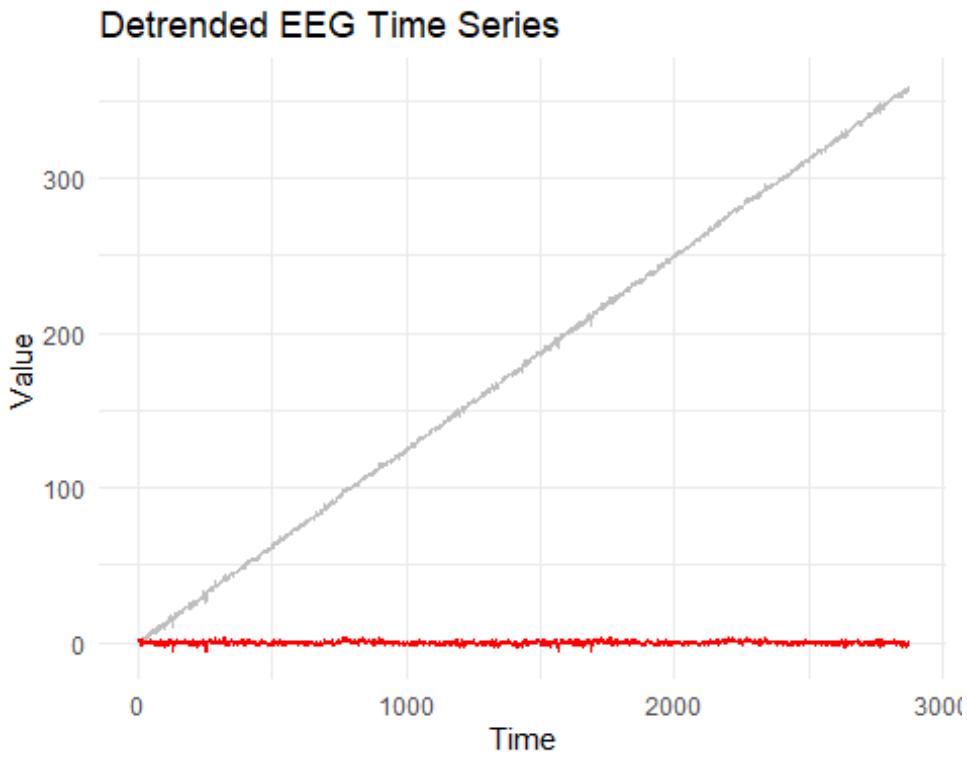
1. Let's check if the data is stationary

```
#Checking if the data is stationary

library(tseries)
kpss.test(df$MeanEEG)

## Warning in kpss.test(df$MeanEEG): p-value smaller than printed p-value

##
##  KPSS Test for Level Stationarity
##
## data:  df$MeanEEG
## KPSS Level = 28.823, Truncation lag parameter = 9, p-value = 0.01

kpss.test(df$Score)

## Warning in kpss.test(df$Score): p-value smaller than printed p-value

##
##  KPSS Test for Level Stationarity
```

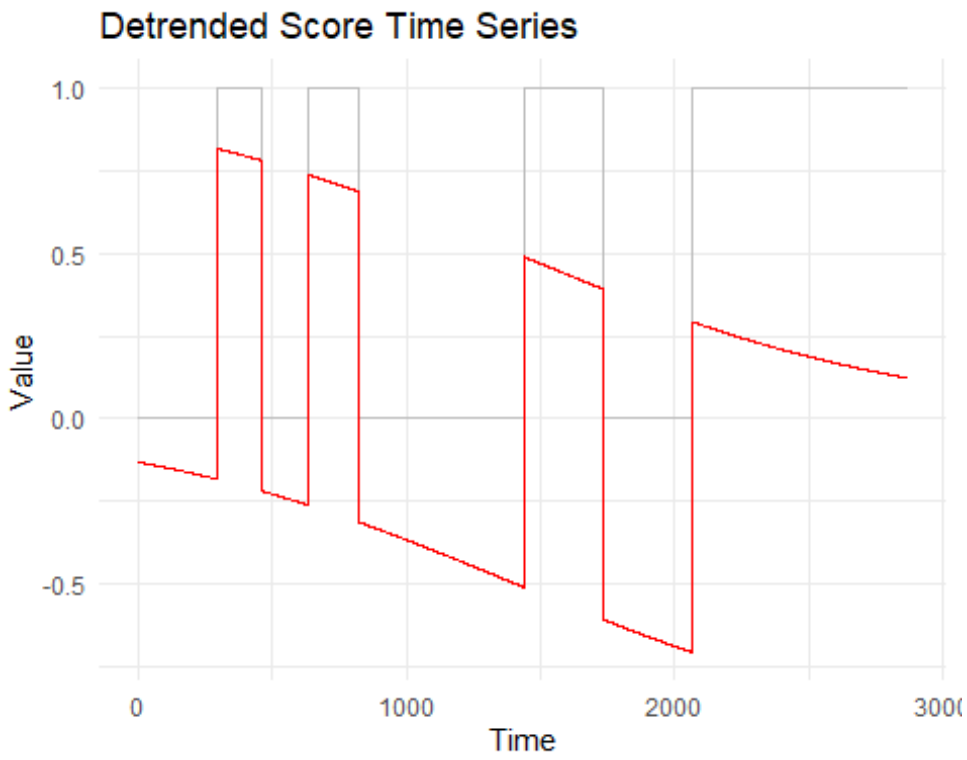
```
##
## data:  df$Score
## KPSS Level = 7.1445, Truncation lag parameter = 9, p-value = 0.01
```

According to the KPSS test, the data for engagement and prediction both seem to not be stationary as the p value is below 0.05. This means we'll have to detrend the data.

```
# Load the required package
library(dplyr)

# Fit a linear regression model to the binary data
model.eeg <- lm(MeanEEG ~ Time, data = df)

# Extract the trend line from the model
trend.eeg <- predict(model.eeg)

# Detrend the data by subtracting the trend line
df$EEGDetrended <- df$MeanEEG - trend.eeg

# Plot the original and detrended data
ggplot(df, aes(x = Time)) +
  geom_line(aes(y = MeanEEG), color = "grey") +
  geom_line(aes(y = EEGDetrended), color = "red") +
  labs(title = "Detrended EEG Time Series", x = "Time", y = "Value") +
  theme_minimal()
```



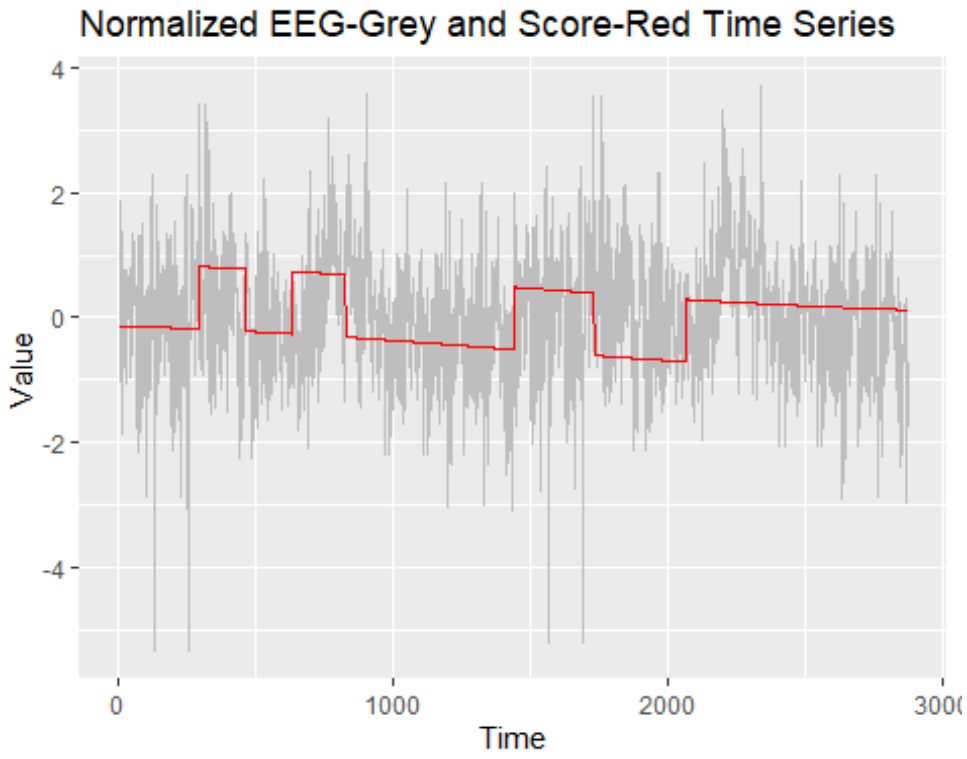
```r
# Load the required package
library(dplyr)

# Fit a logistic regression model to the binary data
model.score <- glm(Score ~ Time, data = df, family = binomial)

# Extract the predicted probabilities from the model
prob.score <- predict(model.score, type = "response")

# Detrend the data by subtracting the predicted probabilities
df$ScoreDetrended <- df$Score - prob.score

# Plot the original and detrended data
library(ggplot2)
ggplot(df, aes(x = Time)) +
  geom_line(aes(y = Score), color = "grey") +
  geom_line(aes(y = ScoreDetrended), color = "red") +
  labs(title = "Detrended Score Time Series", x = "Time", y = "Value") +
  theme_minimal()
```



3. Normalize the data for further empirical dynamic modeling analysis

```r
df$EEGNormalized<- as.numeric(scale(df$EEGDetrended))

plot3 <- ggplot(data=df, aes(x=Time, y=EEGNormalized)) + geom_line(aes(Time,
EEGNormalized),color="grey") + geom_line(aes(Time,ScoreDetrended),
color="red") + xlab("Time") + ylab("Value") + ggtitle("Normalized EEG-Grey
```

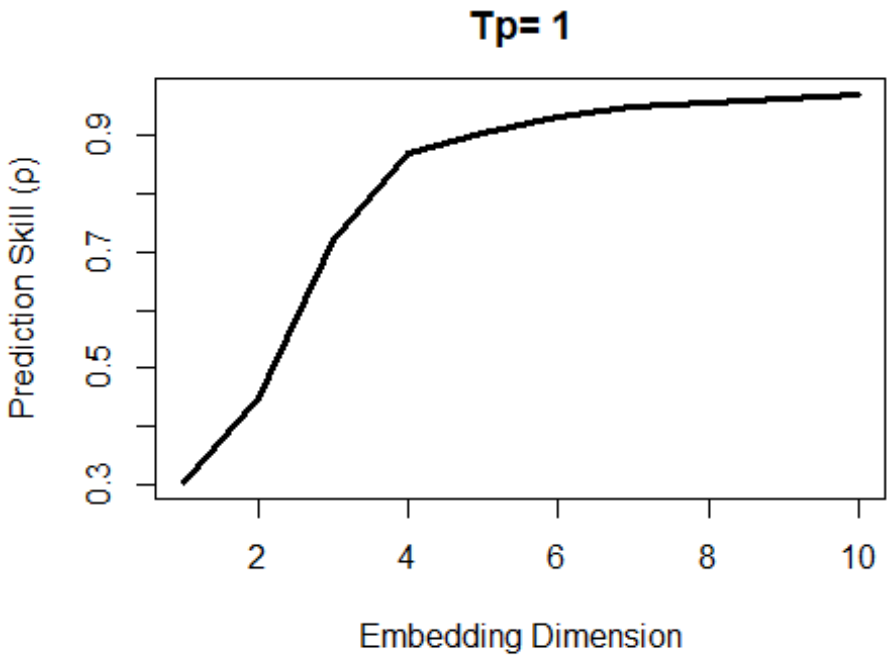
```
and Score-Red Time Series")
plot3
```



Finding Optimal Embedding Dimensions

4. Find the optimal embedding dimension for engagement

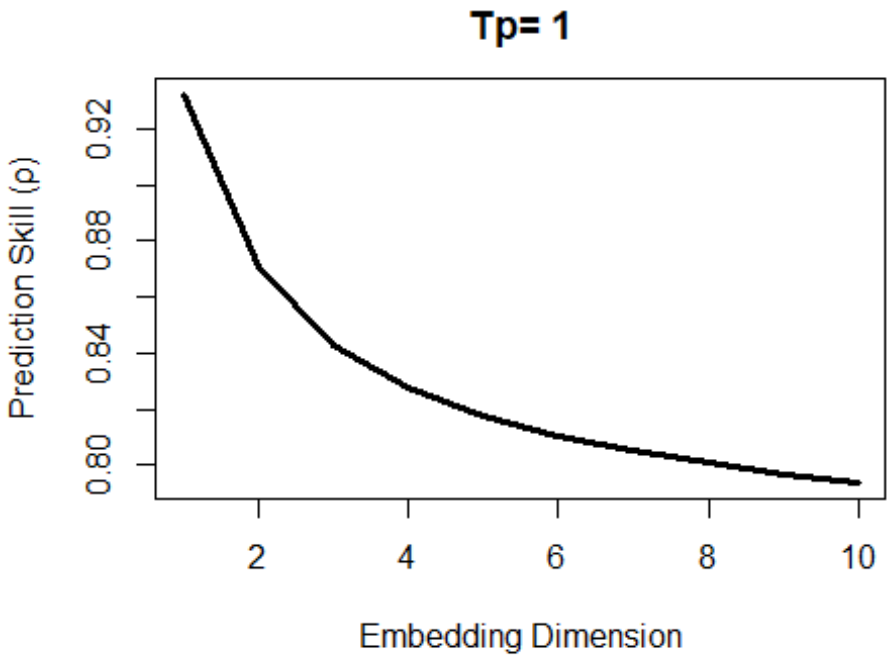
```
# Set library and pred for engagement
lib_point <- c(1,floor(max(length(df$EEGNormalized))/2))
pred_point <-
c(floor(max(length(df$EEGNormalized))/2)+1,max(length(df$EEGNormalized)))

# Check for the embedding dimensions
rho_emd_ML <- EmbedDimension(dataFrame = df, lib = lib_point, pred =
pred_point, columns='EEGNormalized', target ='EEGNormalized')
```



The optimal embedding dimension for engagement seems to be 4, as prediction skill peaks around 4 dimensions.

5. Find the optimal embedding dimension for score

```
# Set library and pred for engagement
lib_point <- c(1,floor(max(length(df$ScoreDetrended))/2))
pred_point <-
c(floor(max(length(df$ScoreDetrended))/2)+1,max(length(df$ScoreDetrended)))

# Check for the embedding dimensions
rho_emd_ML <- EmbedDimension(dataFrame = df, lib = lib_point, pred =
pred_point, columns='ScoreDetrended', target ='ScoreDetrended', Tp = 1)
```



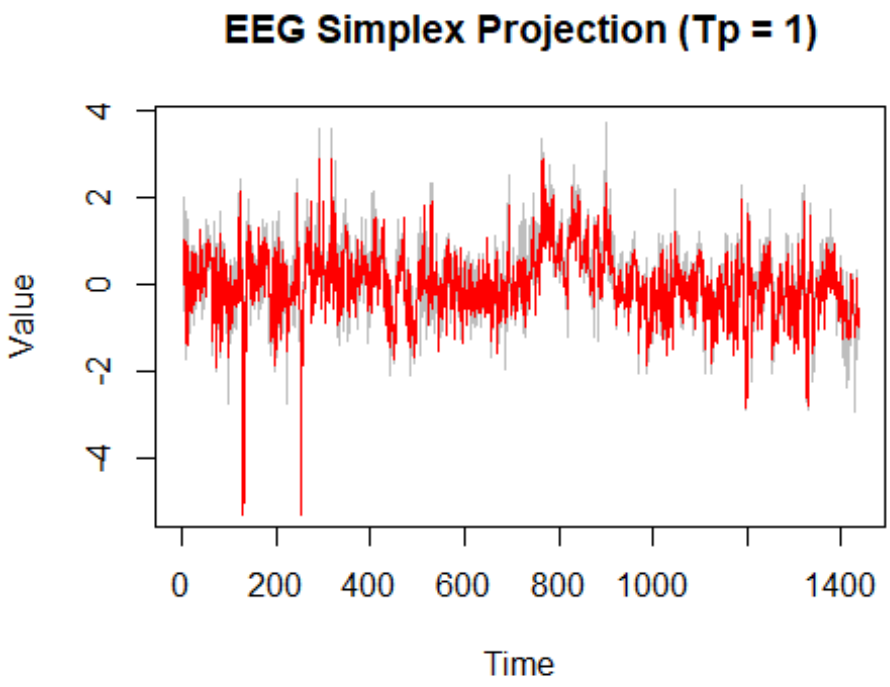
The optimal embedding dimension for score seems to be 1 as prediction skill only goes downhill from there.

Exploring Simplex Projections of EEG Data with Varying Tp Values

1. Let's use the Simplex projection function to compare the observed values against the predicted values for the normalized EEG data.

First let's look at EEG with Tp = 1, meaning the predicted value is one timepoint ahead of the observed value. We'll set E at 4, as indicated by the plot above.

```
simplex_out_ML <- Simplex(dataFrame = df, lib = lib_point, pred = pred_point,
E=4, columns='EEGNormalized', target ='EEGNormalized', Tp = 1)

# Plot observed versus predicted values
plot(simplex_out_ML$Observations,type='l', xlab = "Time", ylab="Value", main
= "EEG Simplex Projection (Tp = 1)", col="grey")
lines(simplex_out_ML$Predictions,type='l',col="red")
```




```
#Let's check the error and $\rho$ using the ComputeError() function.

ComputeError(simplex_out_ML$Observations, simplex_out_ML$Predictions) # Tp =
1

## $MAE
## [1] 0.3709948
##
## $rho
## [1] 0.868539
##
## $RMSE
## [1] 0.5055367
```

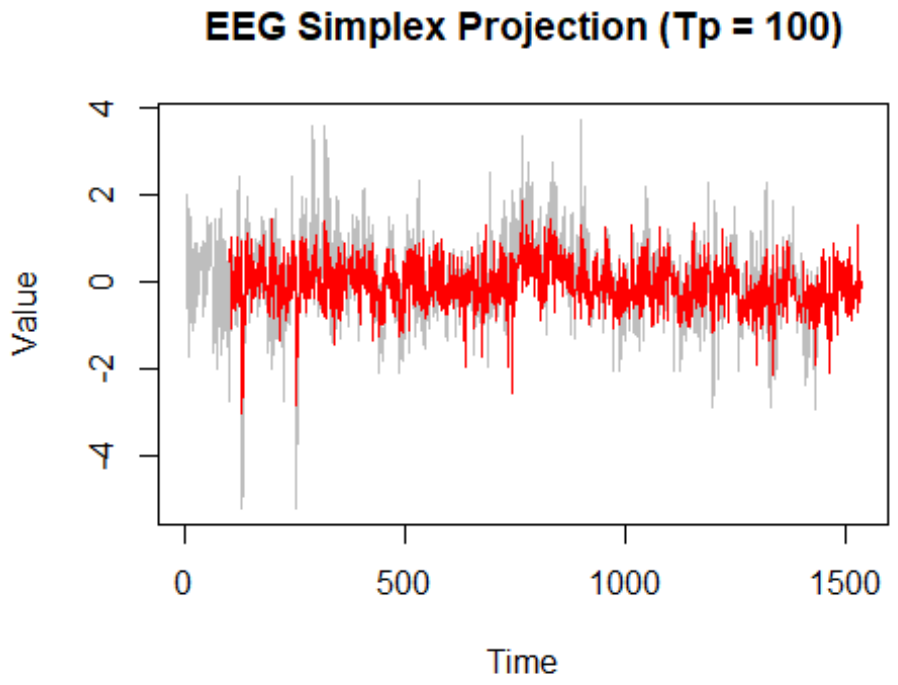
The predicted values are plotted in blue and the observed values are plotted in black. There seems to be a lot of noise, which might make an exact prediction difficult. However, there is stjll a decent amount of overlap that might allow for detecting non-precise trends, such as peaks and drops in engagement values.

The ComputeError() function is used to compute the mean absolute error (MAE), correlation coefficient (rho), and root mean squared error (RMSE) between the observed and predicted values of a time series forecast. A rho of 0.868539 suggests a strong positive correlation, meaning that the model's predictions are generally in line with the actual data. However, the data was sampled 256 times per second, meaning this prediction is only 1/256th of a second in the future.

One of the goals of this system is to eventually be able to predict engagement at longer intervals, so it is also interesting to see how the match between observed and predicted values would change if we increase the value of Tp. Note that Tp signifies the amount of time points separating the observed and predicted value, i.e. how far into the future we are trying to look.

Let's look at the prediction for 100 time points ahead and generate plots and error computations for it.

```
simplex_out_ML2 <- Simplex(dataFrame = df, lib = lib_point, pred =
pred_point, E=4, columns='EEGNormalized', target ='EEGNormalized', Tp = 100)

# Plot observed versus predicted values
plot(simplex_out_ML2$Observations,type='l', xlab = "Time", ylab="Value", main
= "EEG Simplex Projection (Tp = 100)", col="grey")
lines(simplex_out_ML2$Predictions,type='l',col="red")
```

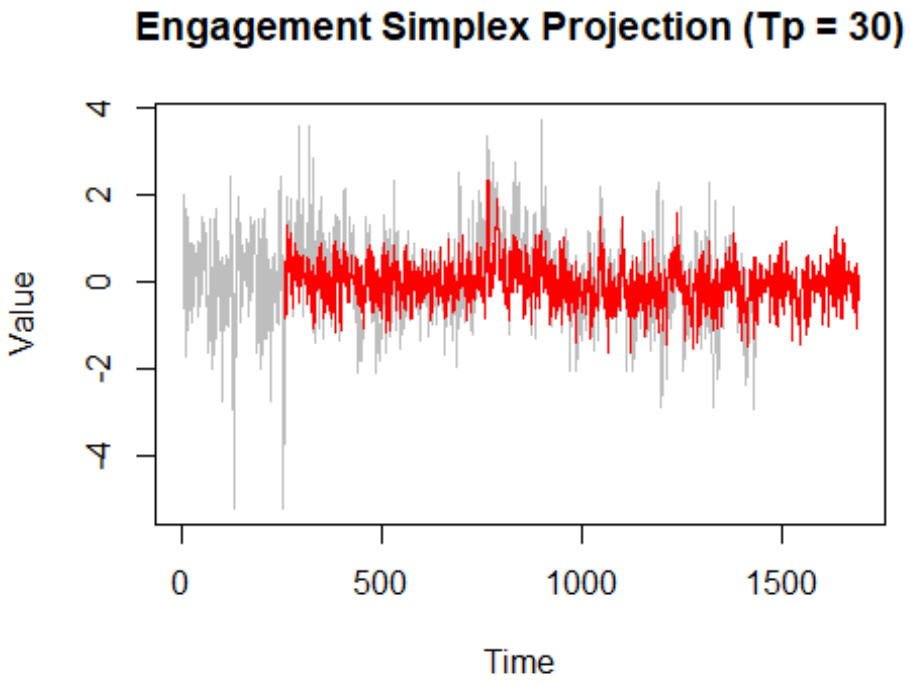



```
#Let's check the error and $\rho$ using the ComputeError() function.

ComputeError(simplex_out_ML2$Observations, simplex_out_ML2$Predictions) # Tp
= 100

## $MAE
## [1] 0.5871803
##
## $rho
## [1] 0.6519688
```

```
##
## $RMSE
## [1] 0.7751556
```

Let's see how it does for one second, because the sampling rate is 256 hZ, that's approximately Tp = 256.

```
simplex_out_ML3 <- Simplex(dataFrame = df, lib = lib_point, pred =
pred_point, E=4, columns='EEGNormalized', target ='EEGNormalized', Tp = 256)

# Plot observed versus predicted values
plot(simplex_out_ML3$Observations,type='l', xlab = "Time", ylab="Value", main
= "Engagement Simplex Projection (Tp = 30)", col="grey")
lines(simplex_out_ML3$Predictions,type='l',col="red")
```




```
#Let's check the error and $\rho$ using the ComputeError() function.

ComputeError(simplex_out_ML3$Observations, simplex_out_ML3$Predictions) # Tp
= 256

## $MAE
## [1] 0.5797914
##
## $rho
## [1] 0.6227007
##
```

```
## $RMSE
## [1] 0.7640916
```

Comparing the three sets of values obtained from the ComputeError() function for different time prediction horizons (Tp):

- simplex_out_ML with Tp=1 has an MAE of 0.371, a correlation coefficient (rho) of 0.869, and an RMSE of 0.506.
- simplex_out_ML2 with Tp=100 has an MAE of 0.587, a correlation coefficient (rho) of 0.652, and an RMSE of 0.775.
- simplex_out_ML3 with Tp=256 has an MAE of 0.580, a correlation coefficient (rho) of 0.623, and an RMSE of 0.764.

Looking at these these results, we can see that the MAE and RMSE values increase as the time prediction horizon (Tp) increases. This is expected, as making longer-term forecasts is generally more difficult and less accurate than making short-term forecasts. In the plots, we can also see there seems to be more disconnect between the observed and predicted values at higher values of Tp, particularly, the predicted values also tend to be less extreme than the observed ones in both directions.

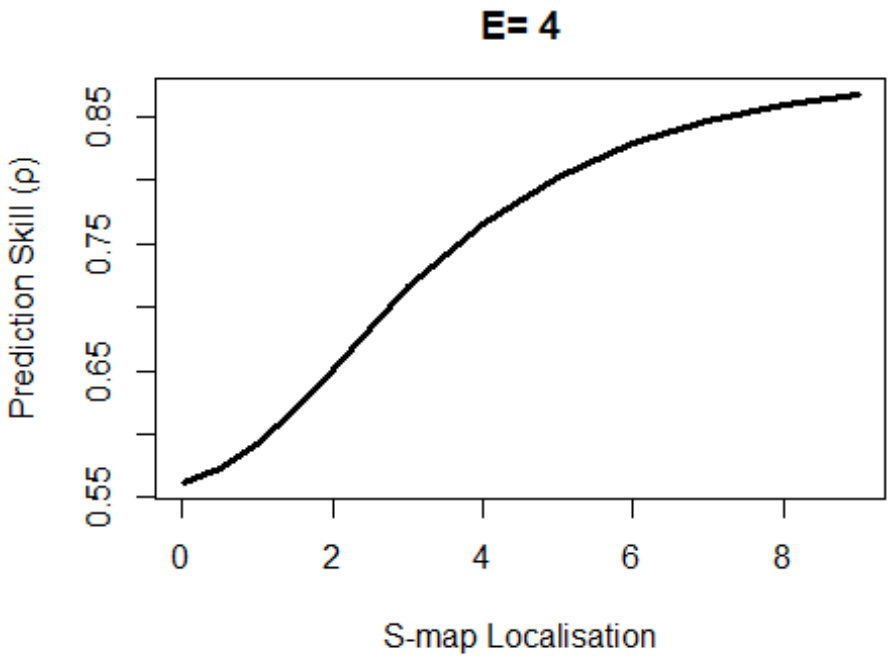
The correlation coefficient (rho) values show a similar pattern. For simplex_out_ML and simplex_out_ML2, the correlation coefficients are positive and moderate, indicating that there is some degree of linear association between the observed and predicted values. However, interestingly, between Tp = 100 and Tp = 256, rho does not decrease significantly and the MAE and RMSE even drop,

Overall, these results suggest that the forecasting performance using the Simplex method is sensitive to the amount of time points between observed and predicted values, and longer-term forecasts seem to be less accurate than shorter-term forecasts but the loss in accuracy does not increase linearly with the amount of Tp as it seems to stabilize after Tp = 100.

Finding the optimal theta value

2. Let's find the optimal theta value for weighting in S-Map projection

```
rho_theta_ML<- PredictNonlinear(dataFrame = df, lib = lib_point, pred =
pred_point, E=4, columns='EEGNormalized', target ='EEGNormalized')
```

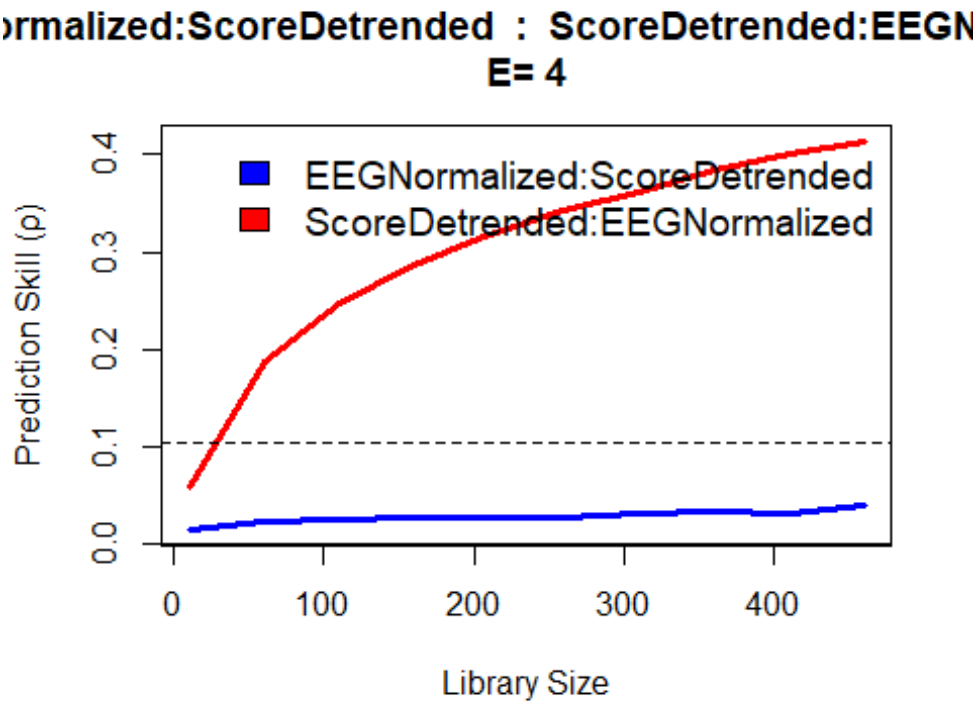


The rho value seems to reach its peak at theta = 10, however, from this plot it is unclear if the prediction skill would keep increasing even further as the value of theta increases. There seems to be a positive linear relationship between theta and rho, which indicates that we are dealing with a non-linear system. As the system is non-linear and can't be fully understood with linear methods, we will try to further our understand of it with empirical dynamic modeling methods.

Convergent Cross-Mapping Analysis of EEG Data vs Recall Scores

1) Is there a bidirectional causal relationship between the EEG data and recall score?

```
# Run a cross correlation
xcor_out <- ccf(df$EEGNormalized,df$Score,lag.max=6,type="correlation",plot =
FALSE)$acf

# Run the convergent cross mapping
cmap <- CCM(dataFrame = df, E = 4, Tp = 0, columns = "EEGNormalized", target
= "ScoreDetrended", libSizes = "10 500 50", sample = 100, showPlot = TRUE )

# Add the cross correlation to the plot
abline(h = max(abs(xcor_out)), col="black",lty=2)
```




```
# Testing for the presence of an increasing monotonic trend
knitr::kable(tidy(Kendall::MannKendall(cmap$`EEGNormalized:ScoreDetrended`)))
```

statistic	p.value	kendall_score	denominator	var_kendall_score
0.8666667	0.0006768	39	45	125

```
knitr::kable(tidy(Kendall::MannKendall(cmap$`ScoreDetrended:EEGNormalized`)))
```

statistic	p.value	kendall_score	denominator	var_kendall_score
1	0.0000831	45	45	125

Does there seem to be a relationship? If so, is it bidirectional or unidirectional?

Both p-values are very small (0.0003467 and 0.0000831), indicating strong evidence of a trend in both EEGNormalized:ScoreDetrended and ScoreDetrended:EEGNormalized. Note that these are interpreted backwards, so ScoreDetrended:EEGNormalized should be interpreted as EEG data driving change in recall scores.

The Mann-Kendall tau statistic measures the strength and direction of the trend. A value of 1 indicates a perfectly increasing trend, -1 indicates a perfectly decreasing trend, and 0 indicates no trend. In this case, the statistic for EEGNormalized:ScoreDetrended is 0.9111111, indicating a strong increasing trend, and for ScoreDetrended:EEGNormalized it's 1, indicating a perfectly increasing trend.

The Kendall score is the sum of the sign of all differences in paired observations. A positive score indicates an increasing trend, while a negative score indicates a decreasing trend. For

both trends, the Kendall scores are positive (41 and 45), further indicating increasing trends.

As both directions of the trend (EEG data driving recall scores and recall scores driving EEG recall) are significantly positive, there seems to be a bidirectional relationship.

However, note that so far we have only analyzed on participant, meaning we can't make any conclusions from this data.

We will proceed with using the analysis performed here as a template and applying it over all participants and then averaging the results to see if the trends observed here hold over the whole dataset (38 participants) and if they are still significant.

Running the analysis on all participants and storing the results

```
library(dplyr)

# Create an empty dataframe to store the results
result_df_random <- data.frame(Participant = integer(),
                         EEGNormalized_ScoreDetrended = numeric(),
                         ScoreDetrended_EEGNormalized = numeric(),
                         stringsAsFactors = FALSE)

# Iterate through participants 3 to 40
for (participant in 3:40) {
  # Subset the base dataframe
  df <- subset(combined_data, Participant == participant & AdaptiveRandom ==
1)
  Time <- 1:length(df$Fz)
  df <- cbind(Time, df)
  df$MeanEEG <- rowMeans(df[, 1:8], na.rm = TRUE)

  # Fit a linear regression model to the binary data
  model.eeg <- lm(MeanEEG ~ Time, data = df)

  # Extract the trend line from the model
  trend.eeg <- predict(model.eeg)

  # Detrend the data by subtracting the trend line
  df$EEGDetrended <- df$MeanEEG - trend.eeg

  # Fit a logistic regression model to the binary data
  model.score <- glm(Score ~ Time, data = df, family = binomial)

  # Extract the predicted probabilities from the model
  prob.score <- predict(model.score, type = "response")

  # Detrend the data by subtracting the predicted probabilities
  df$ScoreDetrended <- df$Score - prob.score
```

```
  # Normalize EEG data
  df$EEGNormalized<- as.numeric(scale(df$EEGDetrended))

  # Run the convergent cross mapping
  cmap <- CCM(dataFrame = df, E = 4, Tp = 0, columns = "EEGNormalized",
target = "ScoreDetrended", libSizes = "10 500 50", sample = 100, showPlot =
FALSE)

  # Store the results in the result dataframe
  result_df_random <- bind_rows(result_df_random, data.frame(Participant =
participant,
                                                             EEGNormalized_ScoreDetrended =
cmap$`EEGNormalized:ScoreDetrended`,
                                                             ScoreDetrended_EEGNormalized =
cmap$`ScoreDetrended:EEGNormalized`,
                                                             stringsAsFactors = FALSE))
}
```

The code above took several hours to run so I will not rerun it for the knitting process but the data were saved in a csv where the Pearson correlation coefficients produced by cmap were saved for EEGNormalized:ScoreDetrended and ScoreDetrended:EEGNormalized. We will look at the results below.

```
result_df<- read.csv("~/Complex Project/results_cmap.csv")

head(result_df)
```

```
##   Participant EEGNormalized_ScoreDetrended ScoreDetrended_EEGNormalized
## 1           3                  0.019933686                    0.1088914
## 2           3                  0.024198886                    0.3436844
## 3           3                  0.012372074                    0.4691934
## 4           3                 -0.002535374                    0.5296259
## 5           3                 -0.010833552                    0.5739504
## 6           3                 -0.026208576                    0.6086504
```

Let's get the mean Pearson correlation coefficient scores to get an idea of the average level of causal effect between EEG data and recall test scores.

```
mean(result_df$EEGNormalized_ScoreDetrended)
```

```
## [1] 0.00507738
```

```
mean(result_df$ScoreDetrended_EEGNormalized)
```

```
## [1] 0.3887258
```

For scores driving change in EEG, the coefficient is very low (0.005), indicating that scores likely do not control EEG data, which makes sense as the EEG data is collected during the memorizing phase and the recall test scores are from a test administered after. In the direction of EEG data driving change in scores, there is a moderately positive correlation

(0.389). Next we'll do a Mann-Kendall trend test to test for the presence of a monotonic trend.

```
# Calculate the average results for EEGNormalized:ScoreDetrended
mean_eeg_score <- mean(result_df$EEGNormalized_ScoreDetrended)
kendall_eeg_score <-
tidy(MannKendall(result_df$EEGNormalized_ScoreDetrended))

# Calculate the average results for ScoreDetrended:EEGNormalized
mean_score_eeg <- mean(result_df$ScoreDetrended_EEGNormalized)
kendall_score_eeg <-
tidy(MannKendall(result_df$ScoreDetrended_EEGNormalized))

# Create a dataframe to store the average results
average_results <- data.frame(Measure = c("EEGNormalized:ScoreDetrended",
"ScoreDetrended:EEGNormalized"),
                              Mean = c(mean_eeg_score, mean_score_eeg),
                              p.value = c(kendall_eeg_score$p.value,
kendall_score_eeg$p.value),
                              stringsAsFactors = FALSE)

# Display the average results using knitr::kable
knitr::kable(average_results)
```

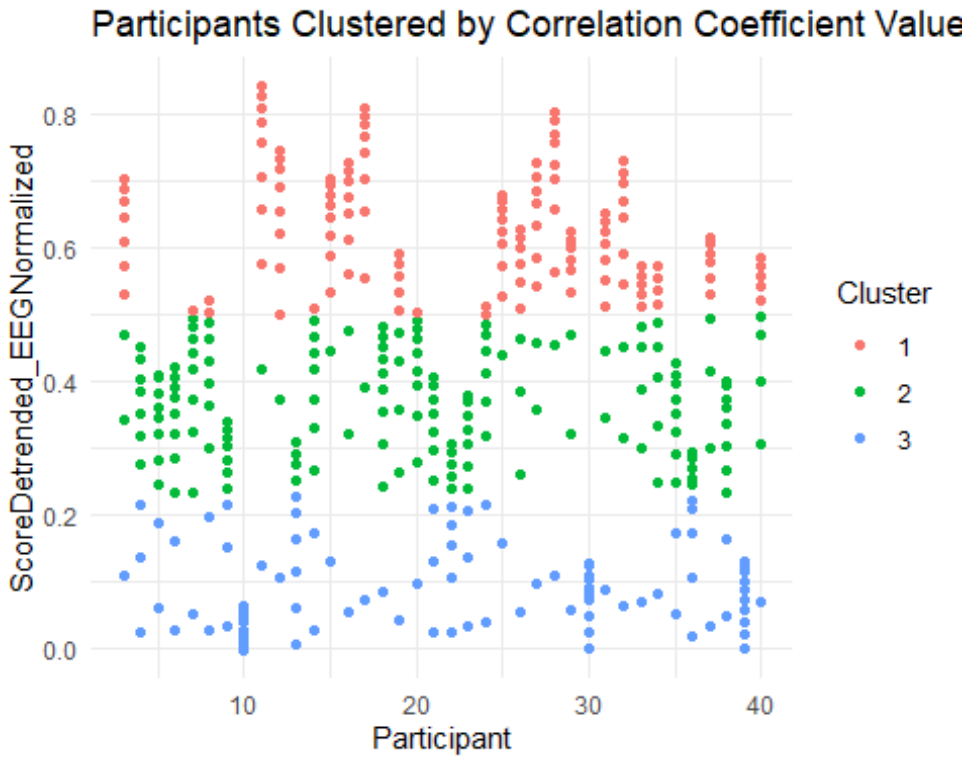
Measure	Mean	p.value
EEGNormalized:ScoreDetrended	0.0050774	0.7893345
ScoreDetrended:EEGNormalized	0.3887258	0.5783665

Neither of the results turned out to be significant, however, a 0.39 correlation coefficient still seems interesting as EEG data are very complicated and predicting or correlating the ability to memorize information or test scores from it has not been done successfully before to my knowledge. When testing with linear methods or known indeces such as the EEG Engagement Index, we found no correlation at all, which makes this result more promising in comparison.

Another thing we have been interested in is clustering participants based on the theory found in literature that different participants will have different EEG metrics (such as the EEG Engagement Index) for different levels of performance in a task and this could be explained by individual factors such as skill. For example, one participant might have a low Engagement index because they are disengaged from the task due to being overwhelmed or unfamiliar with it, leading to low performance, while another might have low engagement due to being overly familiar and bored, which might lead to high performance. Although the approach here is different, let's look at how the participants would cluster based on their correlation coefficients using knn clustering.

```
# Select the required columns for clustering
clustering_df <- result_df[, c("Participant",
"ScoreDetrended_EEGNormalized")]
```

```r
# Perform k-means clustering
k <- 3  # Number of clusters
set.seed(123)  # Set a seed for reproducibility
kmeans_result <- kmeans(clustering_df$ScoreDetrended_EEGNormalized, centers =
k)

# Add the cluster assignments to the dataframe
clustering_df$Cluster <- as.factor(kmeans_result$cluster)

# View the resulting clustering
library(ggplot2)

# Create the scatter plot
ggplot(clustering_df, aes(x = Participant, y = ScoreDetrended_EEGNormalized,
color = Cluster)) +
  geom_point() +
  labs(x = "Participant", y = "ScoreDetrended_EEGNormalized", color =
"Cluster") +
  scale_color_discrete(name = "Cluster") +
  theme_minimal() +
  ggtitle("Participants Clustered by Correlation Coefficient Value")
```



With a k of 3, there are pretty neat clusters of high, medium and low correlation. It could be that the parameters I chose for convergent cross mapping only worked well for the participants in the top end of the plot, as EEG data varies significantly between people. It

could also be that there is something different about these clusters of participants that leads to higher or lower correlations. In future analysis, it could be interesting to look at how these clusters correspond to other information about the participant, such as their overall score and familiarity with the topic or demographic factors such as age, gender and education.

References

- Berkel, N.V., Dennis, S.J., Zyphur, M.J., Li, J., Heathcote, A., & Kostakos, V. (2020). Modeling interaction as a complex system. Human–Computer Interaction, 36, 279-305.
- Chang, C. W., Ushio, M., & Hsieh, C. H. (2017). Empirical dynamic modeling for beginners. Ecological Research, 32(6), 785-796.
- Garcia, C. A. (2022). nonlinearTseries (version 0.2.12) [Computer software]. Retrieved March 20, 2023, from https://github.com/constantino-garcia/nonlinearTseries
- McLeod, A. I. (2022). Kendall (version 2.2.1) [Computer software]. Retrieved March 20, 2023, from https://www.rdocumentation.org/packages/Kendall/versions/2.2.1
- Noakes, L. (1991). The Takens embedding theorem. International Journal of Bifurcation and Chaos, 1(04), 867-872.
- Park, J. (2023). rEDM (version 1.14.0) [Computer software]. Retrieved March 20, 2023, from https://www.rdocumentation.org/packages/rEDM/versions/1.14.0
- Sethi, J.K., & Mittal, M. (2022). Efficient weighted naive bayes classifiers to predict air quality index. Earth Science Informatics, 15, 541-552.
- Wickham, H. (2023). tidyverse (version 2.0.0) [Computer software]. Retrieved February 22, 2023, from https://github.com/tidyverse/tidyverse
- Xie, Y. (2023). knitr (version 1.43) [Computer software]. Retrieved May 25, 2023, from https://www.rdocumentation.org/packages/knitr/versions/1.43
- Couch, S. (2023). broom (version 1.0.4) [Computer software]. Retrieved March 11, 2023, from https://github.com/tidymodels/broom